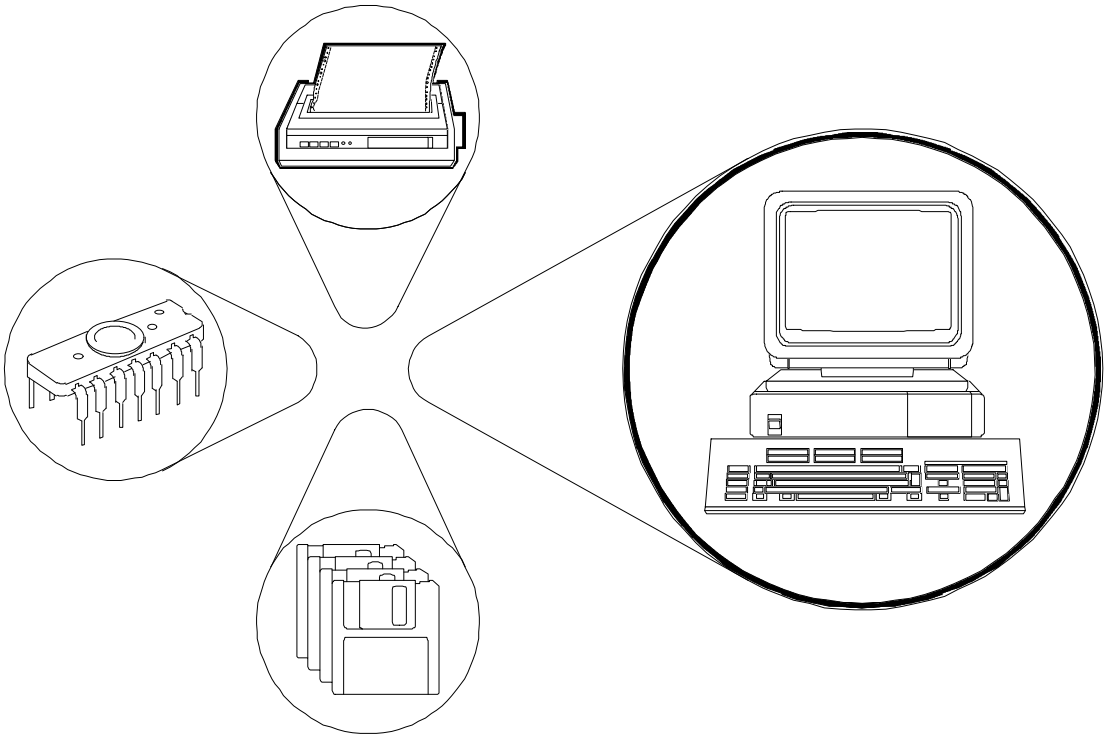
Les systèmes microprogrammés

(Structure de Von Neumann et de Harvard, Processeurs de types C.I.S.C. et R.I.S.C.)



Les systèmes microprogrammés.

Objectifs terminaux.

Ce cours sur les systèmes microprogrammés doit permettre:

- ❖ l'analyse fonctionnelle et la compréhension des structures relatives aux systèmes microprogrammés.
- ❖ la synthèse partielle de ces structures.
- ❖ la détermination de la configuration matérielle minimale, liée aux besoins des entrées et des sorties (capteurs et actionneurs).
- ❖ l'analyse et la compréhension d'un algorithme.
- ❖ la représentation de l'algorithme sous forme d'un organigramme.
- ❖ la traduction de l'algorithme dans le langage adapté à l'unité de traitement.
- ❖ la détermination des méthodes de test et de mise au point, tant sur le plan matériel que logiciel.

I.) Présentation des systèmes microprogrammés.

1.) Avantages et inconvénients des systèmes microprogrammés.

Dans de nombreux cas, les systèmes microprogrammés ont remplacé avantageusement les systèmes en logique combinatoire ou séquentielle. Les progrès sur la rapidité des convertisseurs et des microprocesseurs permettent même l'utilisation des systèmes microprogrammés dans de nombreuses applications analogiques.

La notion de programme enregistré n'est pas nouvelle : Charles Babbage l'avait déjà évoquée en 1833. Ce n'est qu'en 1945, cependant, que John von Neumann (mathématicien américain d'origine allemande), reprenant et améliorant les idées de Babbage, suggère d'incorporer dans la mémoire les instructions de traitement en même temps que les données et selon la même codification. Les instructions sont alors exploitées comme des données ordinaires.

Il a décrit la structure générale des ordinateurs. Cette architecture de machine est encore valable de nos jours, et la plupart des ordinateurs actuels s'en inspirent.

1.1.) L'architecture ou structure de von Neumann:

John von Neumann a proposé une structure universelle de machine à calculer et en a défini les constituants de base.

La machine est composée des éléments suivants:

– un organe de calcul, susceptible d'exécuter les opérations arithmétiques et logiques, l'unité arithmétique et logique (UAL ou ALU) ;

– une mémoire, ou mémoire centrale, servant à la fois à contenir les programmes décrivant la façon d'arriver aux résultats et les données à traiter;

– des organes d'entrée-sortie, ou périphériques, servant d'organes de communication avec l'environnement et avec l'homme;

– une unité de commande (control unit) permettant d'assurer un fonctionnement cohérent des éléments précédents.

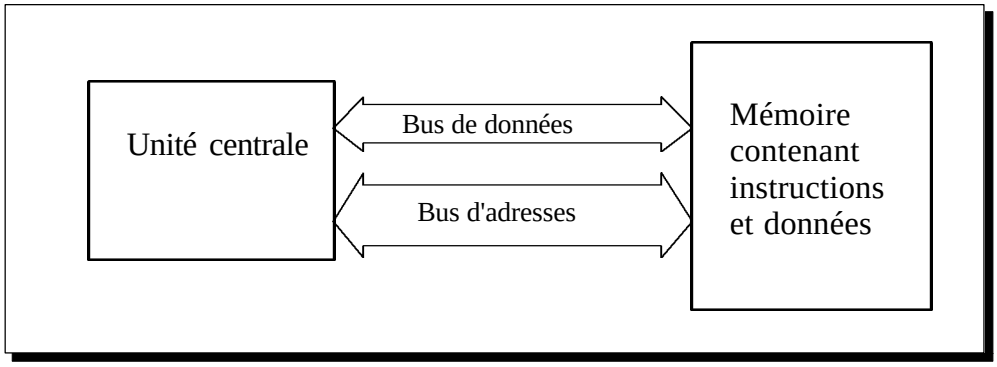
L'ensemble formé par l'unité arithmétique et logique, d'une part, et l'organe de commande, d'autre part, constitue l'unité centrale ou processeur. L'ensemble des composants physiques, appelé matériel (hardware), est commandé par un logiciel (software).

L'unité centrale ne peut effectuer qu'un ensemble restreint d'opérations élémentaires, spécifiées à l'aide d'instructions. L'ensemble des instructions exécutables constitue le jeu d'instructions ; celui-ci caractérise une architecture donnée.

Une instruction est composée de plusieurs parties, les champs , parmi lesquels figurent principalement le code opération , définissant l'opération à exécuter, et l'adresse , précisant la localisation de l'opérande en mémoire.

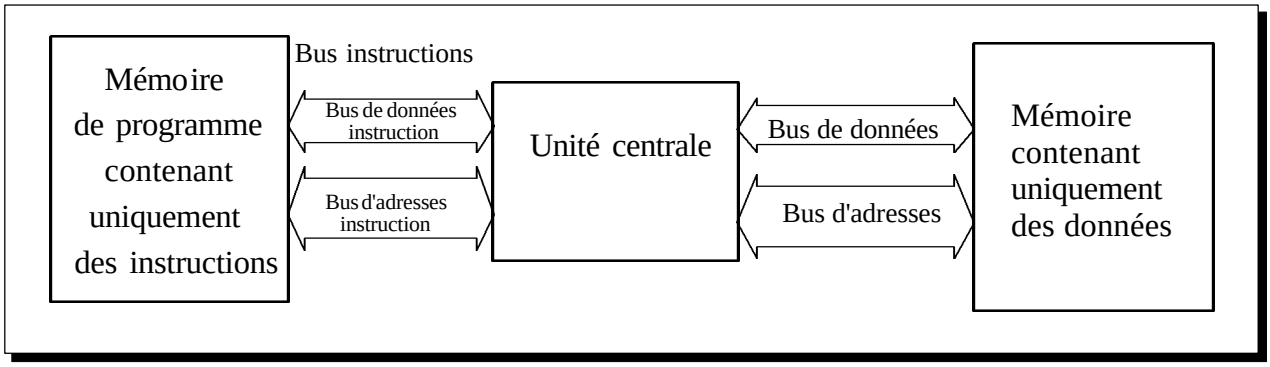
Les instructions de la machine décrite par von Neumann ne comportaient qu'une seule adresse, mais la plupart des machines ultérieures en eurent plusieurs: deux adresses (adresse du premier opérande, adresse du second opérande), voire trois (adresse du premier opérande, adresse du second opérande et adresse de rangement du résultat.

Un microprocesseur basé sur une structure Von Neuman stocke les programmes et les données dans la même zone mémoire. Une instruction contient le code opératoire et l'adresse de l'opérande.



1.2.) Structure de Harvard

Cette structure se distingue de l'architecture de Von Neuman par le fait que les mémoires programmes et données sont séparées. L'accès à chacune des deux mémoires se fait via un chemin distinct. Cette organisation permet de transférer une instruction et des données simultanément, ce qui améliore les performances.



L'architecture généralement utilisée par les microprocesseurs est la structure Von Neuman (exemples : la famille Motorola 68XXX, la famille Intel 80X86). L'architecture Harvard est plutôt utilisée dans des microprocesseurs spécialisés pour des applications temps réels, comme les DSP ou certains microcontrolleurs.

Il existe cependant quelques rares DSP à structure Von Neuman. La raison de ceci est liée au coût supérieur de la structure de type Harvard. En effet, elle requiert deux fois plus de bus de données, d'adresses, et donc de broches sur la puce. Or un des éléments augmentant le coût de productions des puces est précisément le nombre de broches à implanter.

Pour réduire le coût de la structure Harvard, certains DSP ou microcontrolleurs utilisent l'architecture dite « Structure de Harvard modifiée ». À l'extérieur, le DSP ne propose qu'un bus de données et un bus d'adresse, comme la structure Von Neuman. Toutefois, à l'intérieur, la puce DSP dispose de deux bus distincts de données et de deux bus distincts d'adresses. Le transfert des données entre les bus externes et internes est effectué par multiplexage temporel.

1.3.) Types de processeurs (CISC, RISC, ...) :

Les processeurs CISC (Complex Instruction Set Computer) : Architecture des processeurs Motorola 68xxx et de tous les processeurs actuellement présents dans les PC. Les processeurs CISC possèdent plus de 300 instructions internes, contrairement aux processeurs RISC.

Les architectures RISC se caractérisent par un jeu d'instructions limité aux opérations interregistres et aux transferts entre la mémoire et les registres, par l'utilisation de caches mémoires et de pipelines et par un grand nombre de registres. Les instructions sont de longueur fixe pour favoriser l'achèvement de l'exécution d'une instruction à chaque cycle.

C'est au cours des années 80 que la controverse au sujet des processeurs RISC (Reduced Instruction Set Computer: machine à jeu d'instructions réduit) a vraiment commencé.

Les machines microprogrammées des années 1970 offraient des jeux d'instructions très sophistiqués. Au début des années 1980, de nouvelles architectures, les machines RISC (Reduced Instruction Set Computers), ont été conçues de façon à réduire les temps de conception des machines et pour simplifier l'écriture des compilateurs. Par contraste, les machines dotées d'un jeu d'instructions complexe ont été appelées des machines CISC (Complex Instruction Set Computers).

Actuellement, les machines RISC et les machines CISC cohabitent, et aucun des deux types d'architectures n'a réellement supplanté l'autre.

Les processeurs CISC ont vu la taille de leur jeu d'instructions croître de façon exponentielle, ce qui s'est traduit par un microprogramme très complexe. Tant que l'accès à la mémoire centrale était pénalisant par rapport à une lecture en mémoire morte (ROM : Read Only Memory) et un décodage de micro-instructions de plus en plus long, l'architecture CISC était satisfaisante. En outre, la tâche des compilateurs était grandement simplifiée puisqu'ils pouvaient apparier à moindre effort les langages de programmation de haut niveau (Basic, Pascal, C, Fortran, etc.) avec un langage machine très riche.

Or, à cette même période, des études statistiques menées sur les applications ont clairement montré que les programmes générés par les compilateurs se contentaient le plus souvent d'affectations, d'additions et de multiplications par des constantes. L'idée fondatrice de l'architecture RISC n'est donc pas de réduire le nombre d'instructions du processeur - bien que ce dernier soit souvent inférieur à celui d'un processeur CISC -, mais bel et bien de ne retenir que les instructions exécutables en un seul cycle CPU.

Une instruction exécutée en un cycle doit effectuer le chargement des données dans les branches de l'UAL depuis les registres ou la mémoire, puis opérer le calcul et mémoriser le résultat. Il s'agit, comme vous l'avez sûrement remarqué, d'une micro-instruction.

Actuellement, les machines RISC et les machines CISC cohabitent, et aucun des deux types d'architectures n'a réellement supplanté l'autre.

Bien sûr, les implications sont nombreuses, mais s'il ne faut en retenir qu'une ce doit être que, sur une machine RISC, la diminution de la complexité matérielle est compensée par un compilateur très évolué. Windows NT fonctionne sur des processeurs RISC dont les plus connus sont l'Alpha de Digital, et le PowerPC de Motorola. *Avec le futur Merced (renseignements sur internet début 2002), Intel et Hewlett-Packard proposeront un processeur 64 bits hautement parallèle conçu pour le monde Windows (les deux partenaires évoquent une nouvelle architecture baptisée EPIC : Explicitely Parallel Instruction Computing).*

Les langages de haut niveau ont besoin d'être traduits pour être compris par les machines (les programmes de traduction s'appellent des « compilateurs ») et il n'y a dans ce cas aucune raison pour que le langage de l'intérieur même de la machine soit facile à utiliser ; mais il doit être efficace et permettre de bonnes performances de vitesse. C'est ainsi que l'on est arrivé à la notion de machines R.I.S.C. (par opposition au C.I.S.C.), le R. de R.I.S.C., pour reduced , correspondant aussi au fait que l'on s'était rendu compte que beaucoup d'instructions du C.I.S.C. étaient en fait fort peu utilisées.

La quasi-totalité des machines informatiques évoquées jusqu'ici fonctionnent en fait sur le schéma de von Neumann : « Je vais voir dans la mémoire de programmes ce que je dois faire maintenant – Je vais prendre dans la mémoire de données la donnée sur laquelle je dois agir – J'agis – Je retourne voir dans la mémoire de programmes... »

Cette façon de faire, si elle s'est révélée adaptée à la solution de beaucoup de problèmes, n'est pas idéale pour obtenir la rapidité du calcul ; aussi bien, lorsqu'on a besoin de faire toujours le même type de calcul très rapidement, utilise-t-on des machines adaptées, donc plus efficaces.

C'est ainsi qu'en traitement du signal (radar, sonar, parole, etc.) on a besoin de calculer des transformées de Fourier ; on utilisera dans ce cas des calculateurs de structures très différentes (effectuant les calculs en parallèle pour gagner du temps) à base de composants spécifiques dits « papillons » F.F.T. (fast Fourier transform). On parlera parfois pour certaines machines de traitement du signal de machines S.I.M.D. (single instruction multiple data).

C'est ainsi également que lorsqu'un problème de calcul scientifique peut être traité sous forme de calculs vectoriels, on aura avantage à utiliser des calculateurs vectoriels (appelés souvent supercalculateurs).

Les performances de toutes ces machines de pur calcul mathématique ne sont plus exprimées en M.I.P.S. (Million d'instruction per second) mais en Flops (flotting point operations per second), ou plutôt en mégaflops ou en gigaflops (un million ou un milliard de flops).

1.4.) Notion de pipeline

Le principe du pipeline est connu depuis longtemps en informatique, puisqu'il a été utilisé sur les premiers ordinateurs scientifiques. L'exécution d'une instruction passe par plusieurs étapes, exécutées séquentiellement : ***Exemple sur un processeur CISC***:

– Lecture en mémoire de l'instruction à exécuter.

– décodage de l'instruction et lecture en mémoire de l'opérande.

– exécution de l'instruction proprement dite.

Si l'on suppose que les étapes sont de durées égales et que chaque étape est réalisée par un opérateur distinct. Pour permettre l'exécution d'une instruction à chaque cycle, l'ensemble des trois opérateurs forme un pipeline à trois étages. Chaque opérateur correspond à un étage du pipeline. A un instant donné, une instruction (n) est en cours d'exécution, l'instruction suivante (n+1) est en cours de décodage pour une exécution lors du cycle suivant, et l'instruction suivante (n+2) est en cours d'acquisition.

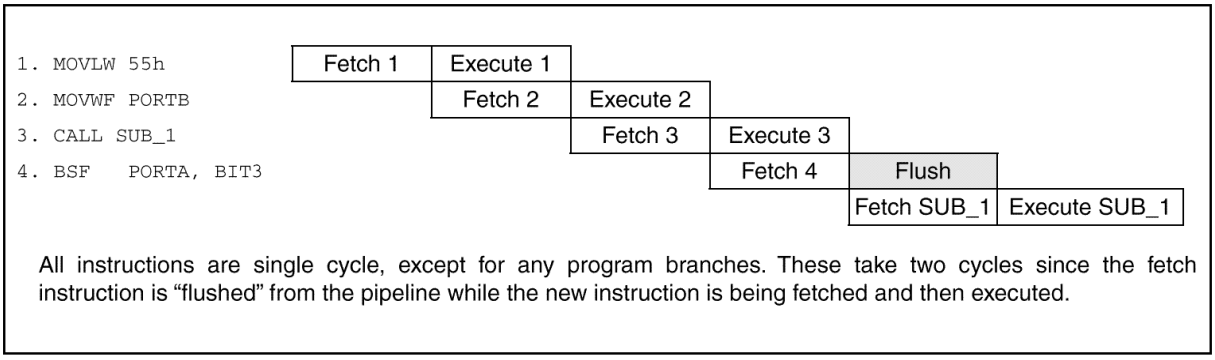
Exemple sur un processeur RISC (Ex: PIC16F84):

– Lecture en mémoire programme de l'instruction à exécuter.

– exécution de l'instruction proprement dite.

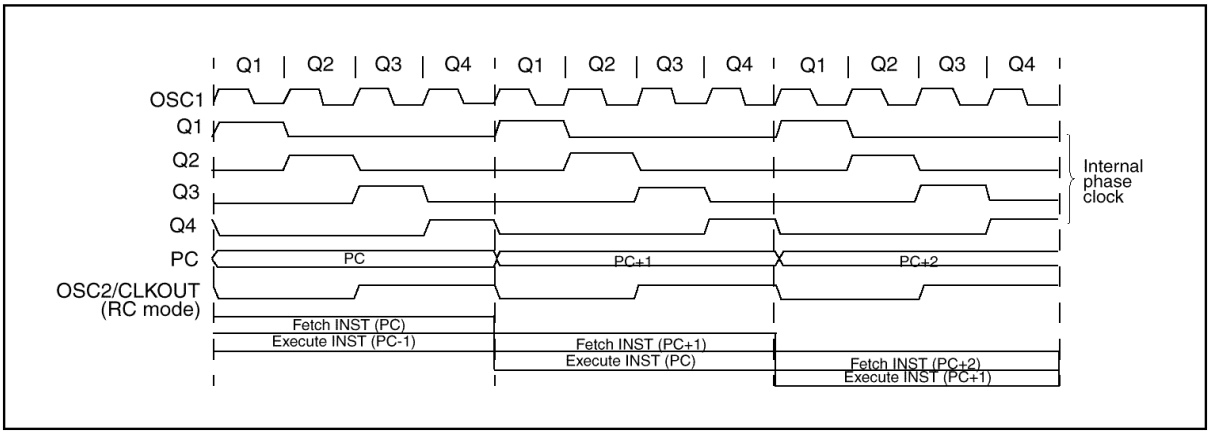
La structure fait appel ici à un pipeline à deux étages. Réduction d'un cycle à l'aide de l' architecture de Harvard (Instruction et opérande disponibles en même temps) et de la structure RISC (instruction traitée en 1 cycle grâce au pipeline). Pendant l'exécution d'une instruction ***("Execute")*** par le deuxième (dernier) étage du pipeline, le premier étage du pipeline procède déjà à l'acquisition de l'instruction suivante ***("Fetch"*** *= aller chercher****)***. Voir figure 3-1 et 3-2 (Extrait de la documentation des PIC16F8X de Microchip).

EXAMPLE 3-1: INSTRUCTION PIPELINE FLOW



All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed.

FIGURE 3-2: CLOCK/INSTRUCTION CYCLE



La conception d'un pipeline est délicate et résulte d'un compromis entre le nombre d'étages qui le compose et la manière dont l'exécution des instructions est décomposée en sous-opérations indépendantes.

1.5.) Les machines superscalaires

Si les machines RISC ont été conçues pour permettre l'exécution d'une instruction par cycle de base, les processeurs superscalaires visent l'objectif encore plus ambitieux d'exécuter plusieurs instructions par cycle.

Pour cela, ces processeurs sont dotés de plusieurs opérateurs arithmétiques pipelinés, chacun d'eux spécialisés dans l'exécution d'un type d'opération (addition de nombres entiers, multiplication de nombres entiers, addition de nombres en virgule flottante, multiplication de nombres en virgule flottante, etc.). Il faut donc acquérir et décoder simultanément plusieurs instructions, afin de lancer leur exécution en parallèle sur les différentes unités. Bien entendu, ce type de machine possède des caches pour permettre au processeur de travailler au maximum de ses possibilités.

L'efficacité d'un processeur superscalaire dépend étroitement du contenu du programme, puisqu'il faut pouvoir alimenter correctement les opérateurs, afin d'exploiter pleinement leurs structures pipelinées (par exemple, le programme doit posséder en nombre suffisant des instructions faisant appel aux différents opérateurs disponibles). Il faut donc disposer de compilateurs réorganisant la séquence d'instructions de façon à exploiter au mieux les ressources matérielles.

Avantages des systèmes microprogrammés sur les systèmes en logique cablé et/ ou en logique programmable (PLD, FPGA):	**Inconvénients:**
1. Conception matérielle plus simple que la plupart des systèmes en logique combinatoire ou séquentielle: système minimum identique pour de nombreuses applications différentes. **2.** Possibilité d'amélioration d'un appareil sans modifier la partie matérielle: amélioration du programme par modification du contenu d'une mémoire. **3.** Facilité d'utilisation par les possibilité d'affichage: mode d'emploi et menu sur afficheur LCD. **4.** La numérisation des informations analogiques permet: une sauvegarde fiable, un traitement et une modification des données, une transmission ou restitution sans perte d'informations. **5.** Réalisation et développement devenus aisés par l'arrivée et le développement rapide de produits performants, à faible coût (Micro-contrôleurs, DSP, DSP-contrôleurs). **6.** La vitesse de traitement ne cesse de croître avec les améliorations technologiques: augmentation des fréquences d'horloge (plus de 100MHz en 1995, 2GHz en 2002), exécution d'une instruction en un minimum de cycle (structures particulières de type RISC et/ou pipe line), utilisation de mémoires rapides SRAM (cache d'instructions et /ou de données). **7.** Progrès considérables sur la rapidité et la résolution des CNAs et CANs permettant le traitement de signaux audio et vidéo en temps réel: aboutissement des recherches sur l'analyse et la synthèse vocale, et sur les images de synthèses.	**1.** Le traitement microprogrammé est (relativement) lent. La vitesse est limitée par le déroulement d'une séquence d'instructions, où chaque instruction élémentaire peut prendre une ou plusieurs périodes d'une horloge de référence. (Rem: des progrès technologiques constants permettent d'améliorer la vitesse de traitement des instructions). **2.** Complexité des cartes électroniques (microprocesseurs avec bus d'adresses et de données de plus en plus grand, mémoires DRAM de grande capacités avec de nombreuses connections (Ex: barrettes 168 broches), périphériques parfois complexes permettant l'accès direct mémoire (DMA). **<u>Rem:</u>** *l'apparition des micro-contrôleurs µC (appelé également mono-chip) a permis d'obtenir des cartes extrêmement simples pour des applications industrielles.*

2. Analyse d'un problème et synthèse par une solution microprogrammée.

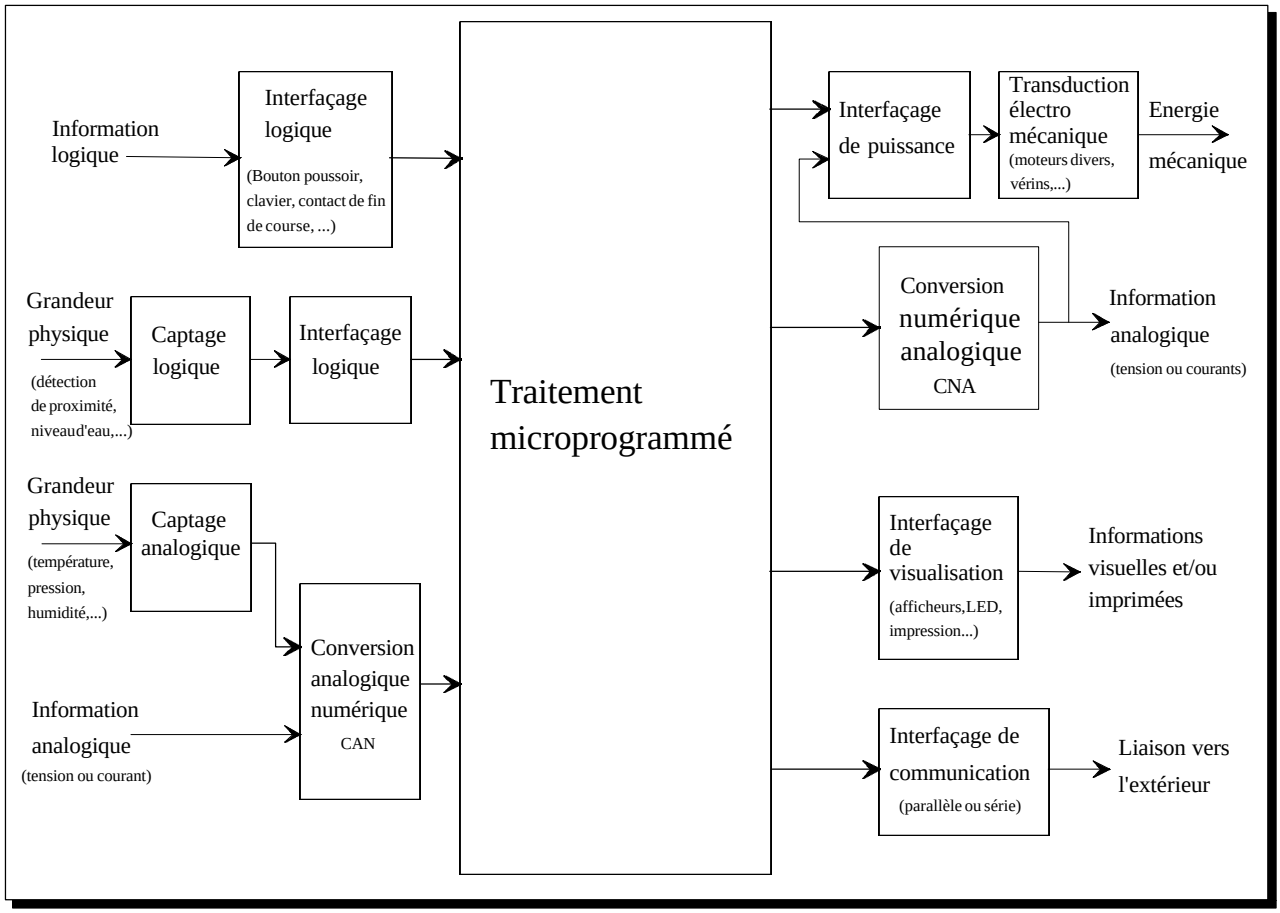
Les systèmes microprogrammés permettent de résoudre aisément tous les problèmes, lorsque la vitesse ("lenteur de traitement") permet leur utilisation. La résolution d'un problème particulier se déroule généralement en 6 étapes:

1ère étape: Détermination de la configuration matérielle.

Après description exacte du problème, en élaborant un cahier des charges, il est possible de déterminer les besoins tant au niveau des entrées (capteurs, interfaces spécifiques), que des sorties (actionneurs, cartes de commande, dispositifs de visualisation et/ou d'impression, dispositifs de communication).

Une analyse fonctionnelle permet d'obtenir la représentation graphique de l'ensemble du système, et permet d'énumérer clairement toutes les fonctions nécessaires. Chaque fonction, ainsi que les entrées et les sorties, devront être caractérisées totalement. (Exemple pour les entrées: Il faut préciser si l'information est analogique, numérique ou alphanumérique, ainsi que le domaine de valeur).

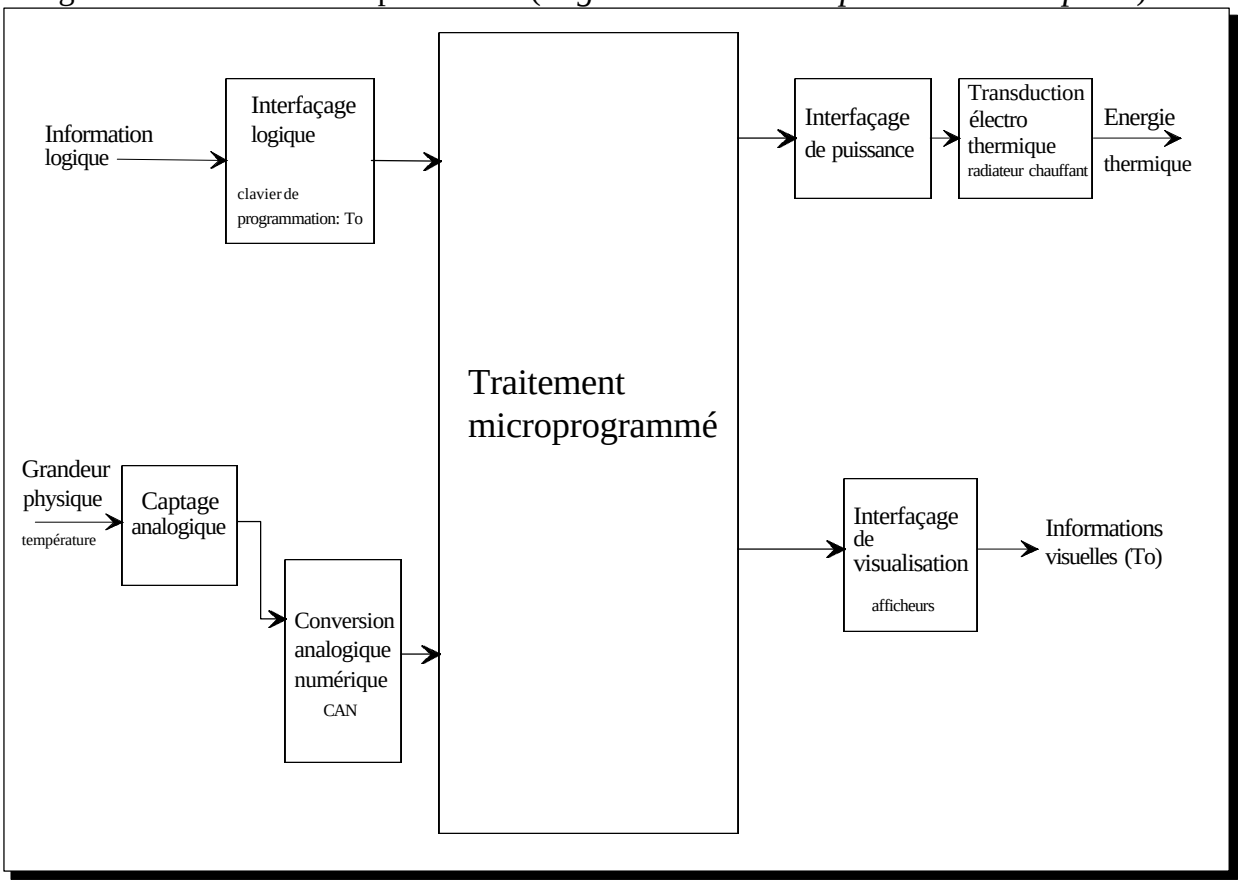
2.1.) Organisation fonctionnelle générale



➪ Illustration sur un problème particulier: ***Régulation de température d'une pièce***

Cahier des charges: La température d'une pièce d'un appartement doit être maintenue à une valeur fixée To, appelée température de consigne, à +/-1°C près (°C = degré celsius). Un appareil de chauffage (radiateur chauffant) est activé ou non, après mesurage de la température de la pièce et comparaison par rapport à To +/- 1.

Organisation fonctionnelle particulière (*Régulation de la température d'une pièce*):



2ème étape: Elaboration d'un algorithme.

L'algorithme consiste en une succession de tâches (appelées également *étapes*), à suivre pas à pas, amenant à la résolution du problème. Il est indépendant du langage de programmation et est écrit en langage naturel. L'algorithme peut être ensuite traduit dans n'importe quel langage ou symbolisme.

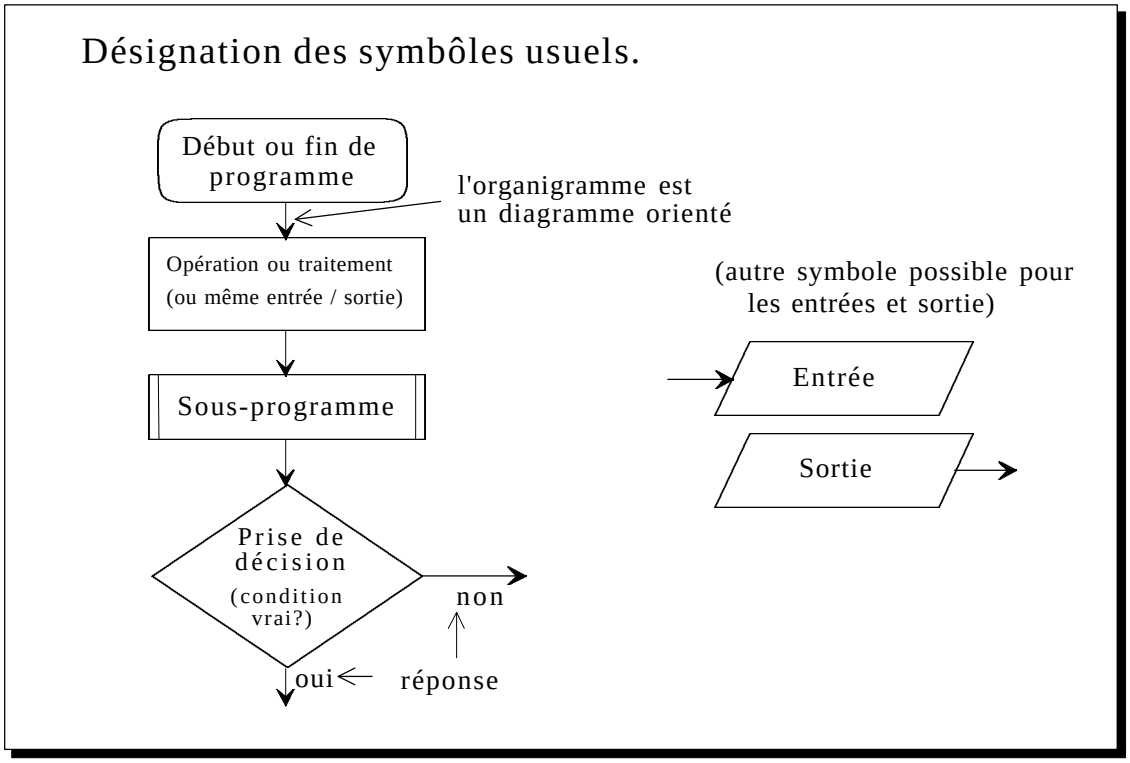
⇨ *Régulation de température d'une pièce:*

Description de l'algorithme

1) Mesurage de la température de la pièce (Tp).
2) Comparaison avec la température limite froide (To -1).
3) Mise en route du chauffage si Tp < To-1 (pièce trop froide).
4) Comparaison avec la limite chaude (To+1).
5) Arrêt du chauffage si Tp > To+1 (pièce trop chaude).
6) Fin de la séquence et retour à la première étape.

3ème étape: Représentation de l'algorithme sous forme graphique: algorigramme ou organigramme de programmation.

Il s'agit d'un symbolisme utilisé pour représenter l'algorithme sous forme graphique.

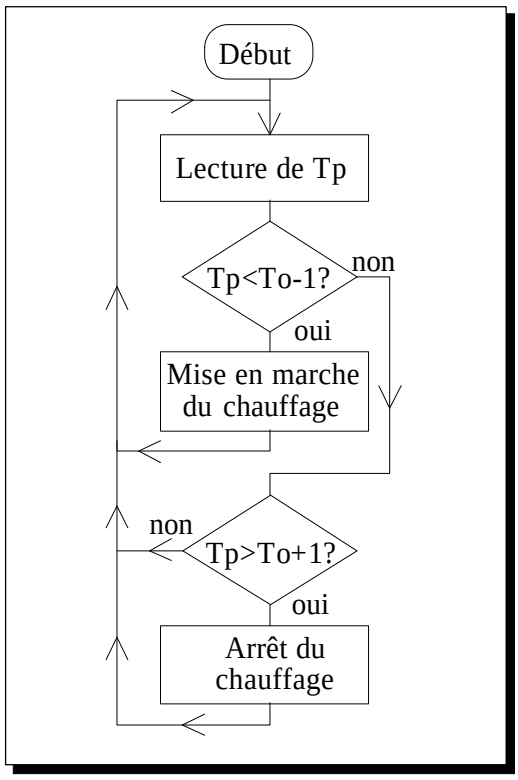


➪ ***Régulation de température d'une pièce:*** Proposition d'un algorigramme

Algorithme correspondant à l'algorigramme ci-contre

(A compléter)

1)

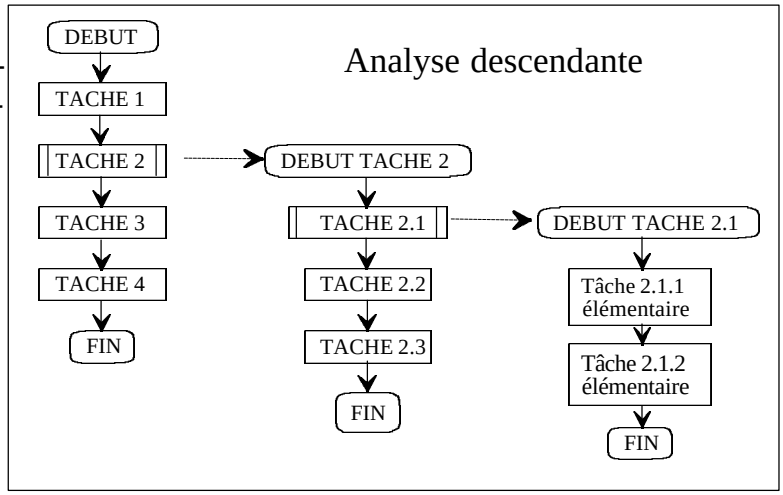


Analysez l'algorigramme proposé de la régulation de température. Vérifiez son adéquation par rapport au problème à résoudre.

Comparez l'algorithme précédemment défini avec cet algorigramme. Proposez l'algorigramme correspondant au précédent algorithme et vice-versa.

Pour des problèmes complexes, la démarche sera **descendante**: les tâches de l'algorithme font appel à d'autres algorithmes.
Cette décomposition des tâches est effectuée jusqu'à l'obtention d'algorithmes connus ou élémentaires.

Ce principe est représenté, dans les algorigrammes, par un symbole différent d'une simple opération. (notion de *sous programme*)

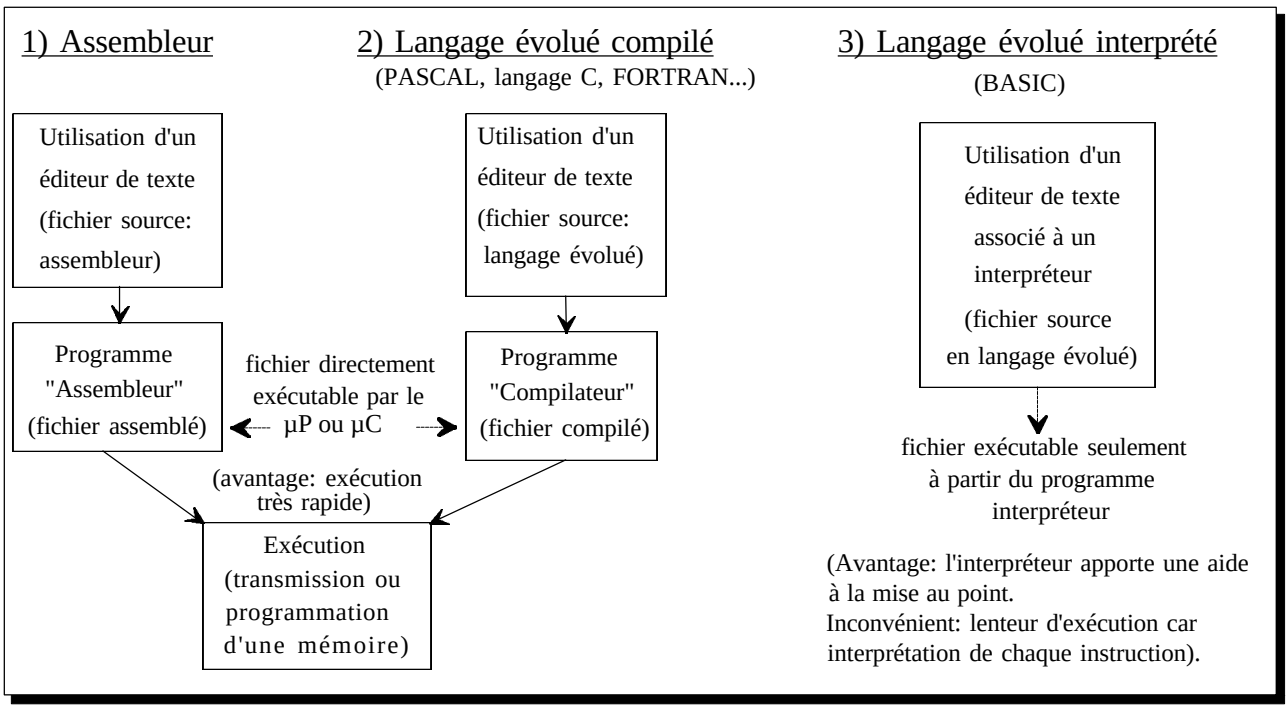


4ème étape: Traduction dans le langage adapté à l'unité de traitement (codage).

Chacune des étapes de l'algorithme sera réalisée par une ou plusieurs instructions dans le langage de programmation choisi.

Le langage compris par les unités de traitement numérique (microprocesseur et microcontrôleur) est le *langage machine*, où chaque instruction se présente sous forme d'un nombre exprimé en binaire (base 2) ou en hexadécimal (base 16).

Le langage machine étant très lourd à utiliser par le programmeur, une représentation sous forme mnémonique (*langage assembleur*) est utilisée. Des langages dits évolués permettent le développement de programmes complexes à l'aide d'interpréteurs (BASIC) ou de compilateurs (PASCAL, langage C, FORTRAN et même BASIC). La mise au point du logiciel est alors facilitée, mais l'exécution est plus lente car les programmes générés sont souvent plus longs.



Remarque: Lorsque le langage choisi est l'assembleur, une décomposition fonctionnelle supplémentaire est très souvent nécessaire pour faciliter le codage. Cette décomposition consiste à détailler l'algorithme (ou l'algorigramme) sous forme graphique (appelé ordinogramme), en tenant compte des caractéristiques de programmation du microprocesseur ou du microcontrôleur choisi.

5ème étape: Test et mise au point du programme.

La mise au point demande souvent une exécution pas à pas du programme (étape par étape). Il est alors possible de vérifier le bon déroulement de la séquence, ou de détecter et de modifier des erreurs dans l'organisation des étapes. L'interpréteur permet souvent un développement facile, par le contrôle permanent des variables et du déroulement du programme.

Les causes de non-fonctionnement du programme sont essentiellement de trois ordres:

- ❖ les erreurs de syntaxe (non-respect des règles d'écriture des instructions dans le langage choisi) dépistées à la compilation
- ❖ les erreurs d'exécution, liées à une connaissance incomplète des instructions utilisées par le langage
- ❖ les erreurs de logique, dues généralement à un algorithme défectueux

Cette phase de développement est très importante et elle met en oeuvre des moyens (logiciels et matériels) conséquents: éditeur de texte, assembleur ou compilateur, simulateur logiciel, émulateur ou programmateur de mémoire, carte microprogrammée cible, appareillage électronique de laboratoire...

6ème étape: Documentation d'utilisation et de maintenance du produit fini.

La documentation doit permettre à tout utilisateur, une recherche rapide des informations. Elle doit donc être structurée: sommaire, présentation, schémas, algorithme et/ou algorigramme (et ordinogrammes, le cas échéant), impression du fichier source (listing du programme), liste des défauts possibles avec méthode de diagnostic et de résolution, index.

II. Organisation des systèmes microprogrammés.

1. Organisation structurelle.

Le coeur d'un système microprogrammé est l'*unité centrale de traitement* (UC ou CPU "Central Process Unit", MPU "MicroProcessing Unit", ou plus simplement *microprocesseur* µP).

✗ Le microprocesseur se compose essentiellement d'une **unité de commande** (séquenceur) qui décode et assure l'exécution des instructions du programme (transfert de données, saut conditionnel, appel à un sous-programme...), et d'une **unité arithmétique et logique** (UAL ou ALU, en anglo-saxon) qui exécute les opérations arithmétiques (addition, soustraction...) et logiques (fonctions booléennes: ET, OU...) en relation avec des registres (cases mémoires internes servant à recevoir les données et les résultats).

✗ Pour permettre l'exécution du programme, il faut que la suite d'instructions (en langage machine) soit conservée dans une mémoire. Il existe 2 types de mémoires:

❖ Les *mémoires mortes* (ROM: Read Only Memory): les informations y sont écrites une seule fois, elles sont alors permanentes, même en cas de coupure d'alimentation (arrêt de la machine ou coupure accidentelle EDF). Une fois programmée par un programmateur, ces mémoires comportent le programme résident.

Il existe différentes technologies de mémoires mortes, dont certaines sont effaçables puis reprogrammables (UV-EPROM: effaçable par UV, EEPROM: effaçable électriquement).

❖ Les *mémoires vives* (RAM: Random Access Memory): les informations peuvent être lues ou écrites autant de fois que désiré, mais elles sont volatiles (toutes les informations sont perdues en cas de coupure de l'alimentation). Ces mémoires sont surtout utilisées pour les données (parfois pour les programmes lorsque ceux-ci ne peuvent pas être résidents). Les données sont toutes les variables provenant de l'environnement extérieur (les variables d'entrée) et les variables générées lors de l'exécution du programme (résultats intermédiaires et résultats finaux représentant les variables de sortie). Il existe également différentes technologies de mémoires vives: SRAM = RAM statique souvent très rapide mais couteuse donc réservées aux faibles capacités (micro-controleurs, mémoire cache, appareils de mesure oscilloscopes à mémoire numérique. DRAM = RAM dynamique souvent de grande capacité pour un coût modeste. Elles sont plus lentes que les SRAM et nécessitent un rafraichissement (=> perte de temps). De multiples versions de DRAM existent liées aux évolutions téchnologiques.

Les mémoires (mortes ou vives) sont assimilables à de grands tableaux, ayant en général huit colonnes (qui forment un *octet*). Les lignes (ou cases mémoires) sont numérotées. Ce numéro représente l'*adresse* de la case mémoire.

Exemple: Circuit UV-EPROM 2764

Cette mémoire morte effaçable par UV est organisée en un tableau de 8 x 1024 lignes de 8 bits, soit 8 kilo-octets (8 x 8 = 64).

✗ Les variables d'entrée et de sortie du système microprogrammé sont véhiculées par des circuits d'interface de périphérie, appelés *circuits périphériques*. Les circuits périphériques permettent le dialogue avec l'environnement et sont de nature très variée.

❖ Les *ports parallèles d'entrée/sortie* (appelés PIA, PIO ou PPI selon les constructeurs de circuits microprogrammés: MOTOROLA, ZILOG, INTEL...). Ils permettent, par exemple, la réception d'informations en provenance d'un clavier et la sortie d'informations vers une imprimante parallèle.

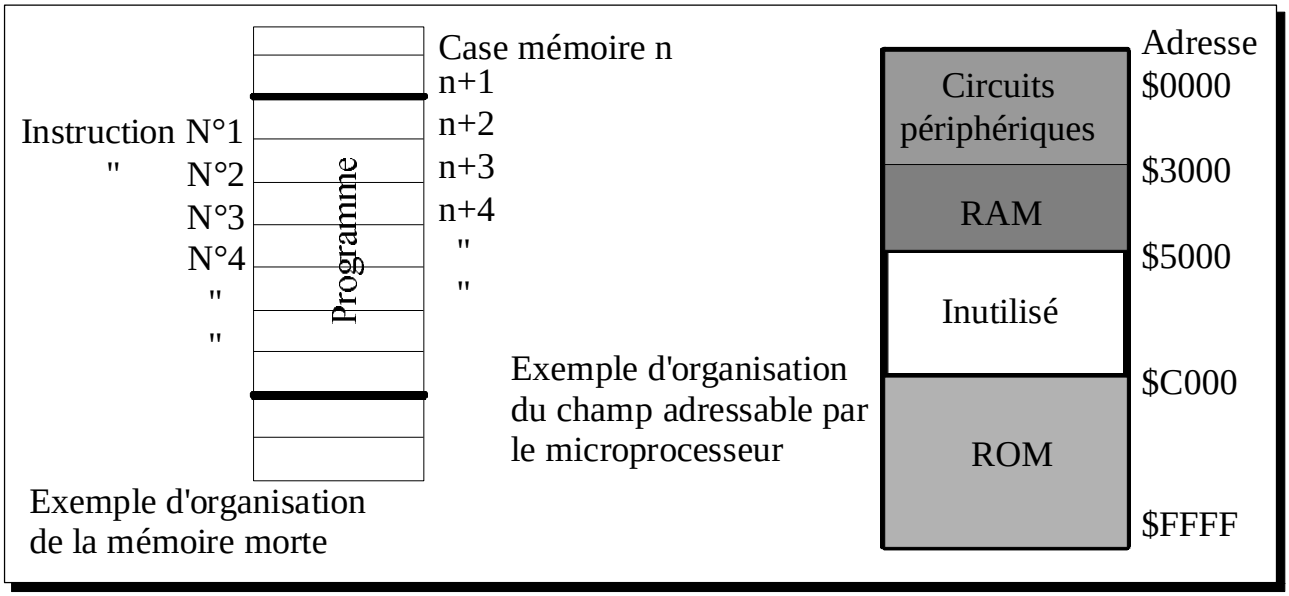
❖ Les *ports sériels* permettant la communication en liaison série synchrone ou asynchrone (communication à distance via le réseau téléphonique et des MODEMs). Ces interfaces sont appelées ACIA, UART, SSDA selon les constructeurs.

❖ Les *circuits de comptage* (Timer, PTM...), qui servent au comptage d'événements extérieurs ou à la génération de signaux.

❖ Les circuits de conversion d'information (CAN et CNA, compatibles microprocesseur), qui servent d'interface entre le domaine analogique et le domaine numérique.

❖ et bien d'autres circuits (Contrôleur d'écran CRTC, Contrôleur de disques FDC, Contrôleur d'interruption PIC, contrôleur d'accès direct mémoire DMA...)

Ces circuits périphériques contiennent des registres internes adressables par le microprocesseur, de façon analogue aux mémoires. Pour que le microprocesseur puisse s'adresser sans ambiguïté à la mémoire morte (programme résident) ou à la mémoire vive (stockage des variables) ou à l'un des circuits périphériques de l'application, il faut que chacun de ces circuits ait une adresse différente dans le champ adressable par le microprocesseur.



✗ Le microprocesseur dialogue avec les mémoires et les circuits périphériques aux moyens d'un ensemble de fils de liaison, appelé *bus de communication*. Il existe trois types de bus dans un système microprogrammé.

❖ Le *bus d'adresses*. Le microprocesseur impose sur ce bus, l'adresse de la case mémoire ou du registre, avec qui il veut dialoguer. Ce bus est donc unidirectionnel, car seul le microprocesseur fournit des adresses. Le nombre de fils qui composent le bus d'adresses, détermine la capacité de la mémoire qui peut être reliée au microprocesseur.

Exemple: Famille 6809 de MOTOROLA

Le bus d'adresses est de 16 fils d'adresses (A0 à A15). Par conséquent, le champ mémoire adressable par le microprocesseur est de 2^{16} cases mémoires (65536 cases mémoires), de l'adresse $0000 à l'adresse $FFFF ($ signifie "hexadécimal", notation MOTOROLA).

❖ Le *bus de données*. Ce bus véhicule les données entre le microprocesseur et les mémoires ou circuits périphériques. Le bus de données est donc bidirectionnel, car le microprocesseur peut lire ou écrire une donnée dans ces circuits. La taille du bus de données permet une classification des microprocesseurs.

Exemple: Famille 6809 de MOTOROLA

Le bus de données est de 8 bits (octet). Le microprocesseur est dit "µP 8 bits".

Famille 68000 de MOTOROLA

Le bus de données est de 16 bits (Word). Le microprocesseur est dit "µP 16 bits".

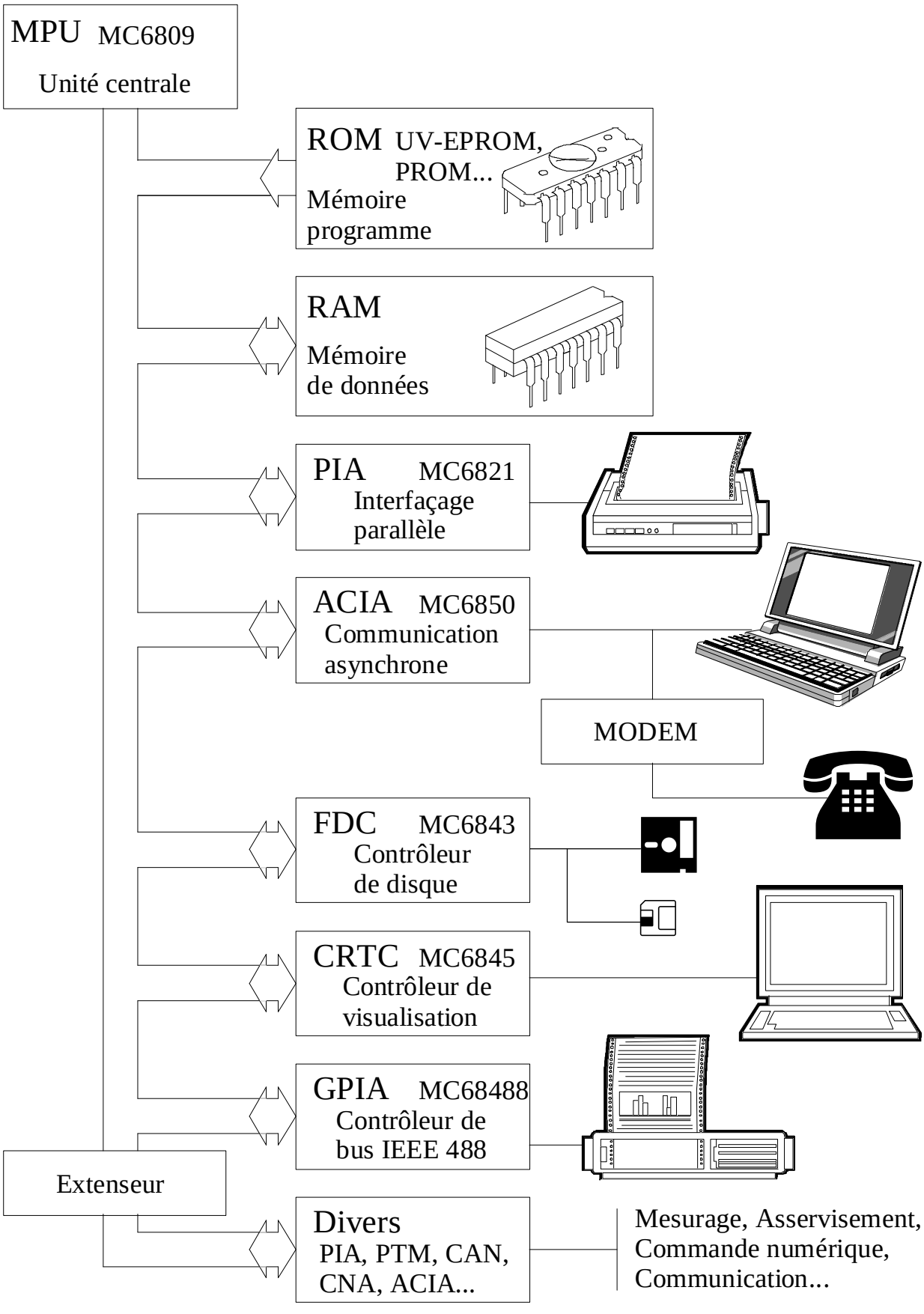
❖ Le *bus de contrôle*. Ce bus comporte un nombre variable de fils suivant le microprocesseur choisi. Il est principalement constitué de signaux, appelés *signaux de contrôle*, qui permettent la gestion et la synchronisation des échanges entre le microprocesseur et les mémoires ou circuits périphériques. Comme par exemple, l'indication de l'instant et du sens de transfert des données par R/$\overline{W}$, et E (MOTOROLA) ou $\overline{RD}$ et $\overline{WR}$ (INTEL).

Une partie du bus de contrôle et du bus d'adresses est exploitée par une fonction, appelée *Décodage d'adresses*, afin de ne sélectionner que le circuit (mémoire ou circuit périphérique) qui est autorisé à dialoguer avec le microprocesseur à un instant donné.

Une illustration de l'architecture d'un système microprogrammé figure en page suivante.

Architecture générale d'un système microprogrammé

Illustration de la gamme microprogrammée 8 bits de Motorola



Travail à effectuer (TD)

Un exemple de schéma structurel d'un système microprogrammé, relatif à un système technique appelé Vibroséis, figure page 16 à 18.

1) Repérez et identifiez sur ce schéma structurel l'unité centrale, les mémoires, les circuits périphériques et la structure de décodage d'adresses.

2) Repérez les bus microprocesseurs (bus de données, bus d'adresses et bus de contrôle). Indiquez le format de ces bus (nombre de fils utilisés pour chacun des bus).

3) Identifiez la taille (exprimée en kilo-octets) de la mémoire programme et de la mémoire de données. Analysez la relation liant la capacité d'une mémoire (sa taille en ko) au nombre de fils d'adresses utilisés par le circuit.

4) Rappelez fonctionnellement le rôle de la structure de décodage d'adresses. Analysez succinctement le schéma structurel relatif à cette fonction, et complétez la feuille réponse (tableau "Carte d'adressage", et la Cartographie mémoire "Memory Map").

5) Aidez vous des fichiers VHDL suivants (décodage d'adresses) et proposez deux fichiers en VHDL sous forme d'équations logiques puis de description comportementale , afin de remplacer les circuits logiques réalisant le décodage d'adresses par un PAL ou GAL. Proposez alors un PAL ou GAL susceptible de contenir la fonction décodage.

```
-- VHDL created by OrCAD Express
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

ENTITY PLD_VHDL is
        PORT (
                CSRom1_Bar: OUT STD_LOGIC;
                CSRom2_Bar: OUT STD_LOGIC;
                E       : IN STD_LOGIC;
                A13     : IN STD_LOGIC;
                A14     : IN STD_LOGIC;
                A15     : IN STD_LOGIC);
END PLD_VHDL;

ARCHITECTURE behavior OF PLD_VHDL IS
BEGIN
CSRom1_Bar <= not ( E and A15 and A14 and A13 );
CSRom2_Bar <= not ( E and A15 and A14 and not A13 );
END behavior;;
```

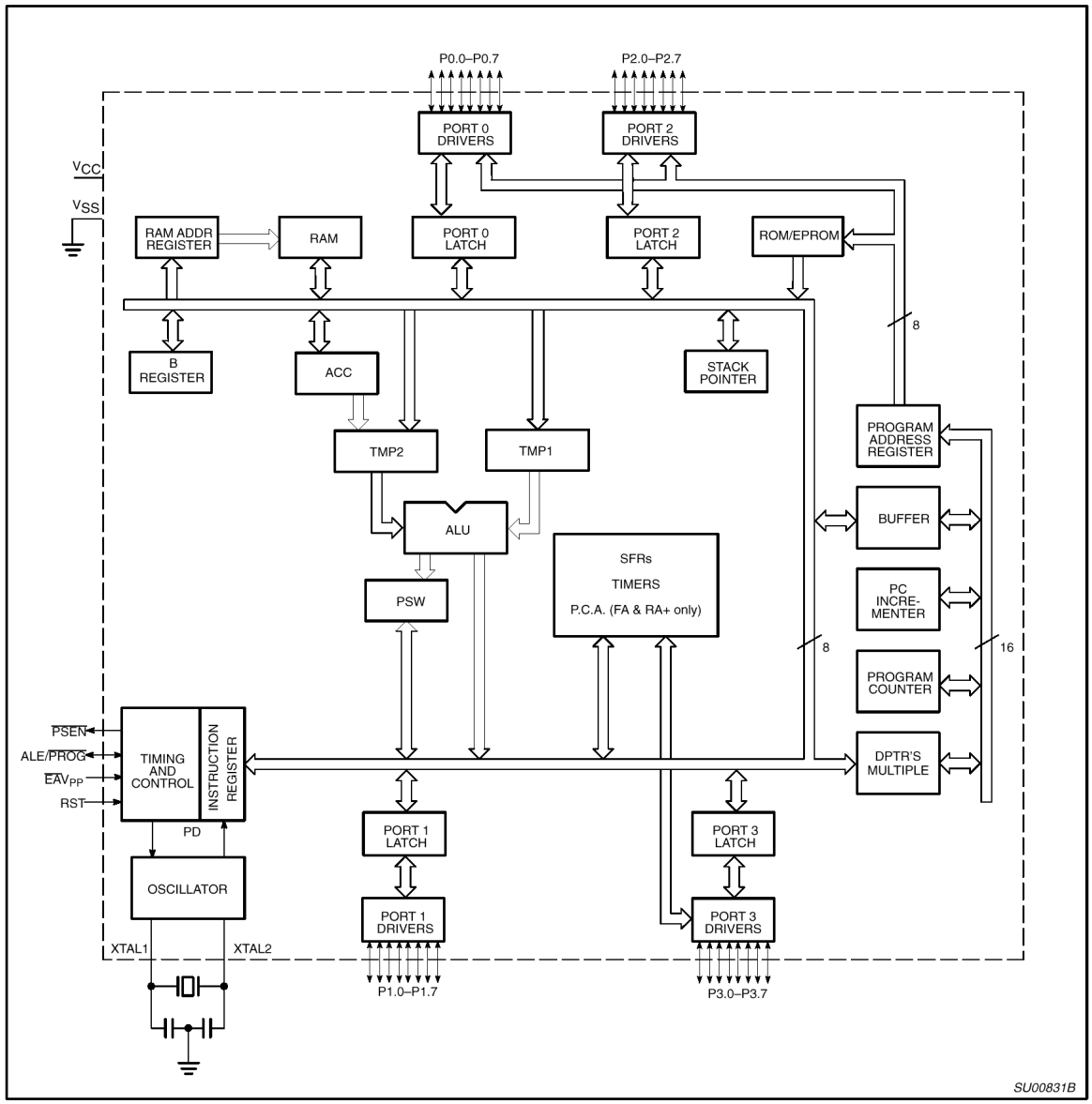
```
-- VHDL created by OrCAD Express
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

ENTITY \Decodag\ is
   PORT (
        PIACS_Bar : OUT STD_LOGIC;
        TIMCS_Bar : OUT STD_LOGIC;
        RAM_Bar   : OUT STD_LOGIC;
        EPROM_Bar  : OUT STD_LOGIC;
        ADR       : IN unsigned (15 DOWNTO 0);
        E         : IN STD_LOGIC;
        RW        : IN STD_LOGIC);
END \Decodag\;

ARCHITECTURE behavior OF \Decodag2\ IS
BEGIN
   EPROM_Bar <= '0'  when (E='1' and RW='1' and ADR >= X"C000" and ADR <= X"FFFF") else '1';
   RAM_Bar <= '0'  when (E='1' and ADR >= X"2000" and ADR <= X"3FFF") else '1';
   TIMCS_Bar <= '0' when (E='1' and ADR >= X"0040" and ADR <= X"0047") else '1';
   PIACS_Bar <= '0' when (E='1' and ADR >= X"0048" and ADR <= X"004F") else '1';
END behavior;
```

✗ Afin d'améliorer la fiabilité et diminuer le coût des systèmes micoprogrammés, les constructeurs se sont orientés, à partir de la fin des années 1970, vers une intégration des circuits, sur une même puce de semi-conducteur. Ces nouveaux composants s'appellent des *microcontrôleurs* (apparition au début des années 1980 des microcontrôleurs 8048 d'INTEL et 68HC05 de MOTOROLA). La tendance actuelle est caractérisée par une diversification très importante de produits au sein d'une même famille de microcontrôleur, afin de multiplier les applications possibles.

Les applications sont par exemple, dans le domaine "grand public", liées au monde de l'audiovisuel (téléviseur, magnétoscope, camescope...): télécommande, incrustation d'image, gestion des réglages, télétexte, décodeur. Mais également, les calculatrices, les traducteurs et dictionnaires électroniques de poche, les appareils ménagers, ou encore les appareils photos, intègrent un microcontrôleur. Dans le monde industriel, les microcontrôleurs sont utilisés dans les applications de contrôle de machines outils ou de robots, dans le pilotage de petits périphériques informatiques (clavier, souris, scanner...), dans les systèmes de télécommunication (téléphone portable), dans l'automobile (contrôle des sièges, alarme, climatisation, injection directe, ordinateur de bord, régulateur de vitesse, ABS...).

Exemple N°1: Microcontrôleur 8051 d'INTEL (doc Philips 8XC51, 3030.pdf).

BLOCK DIAGRAM

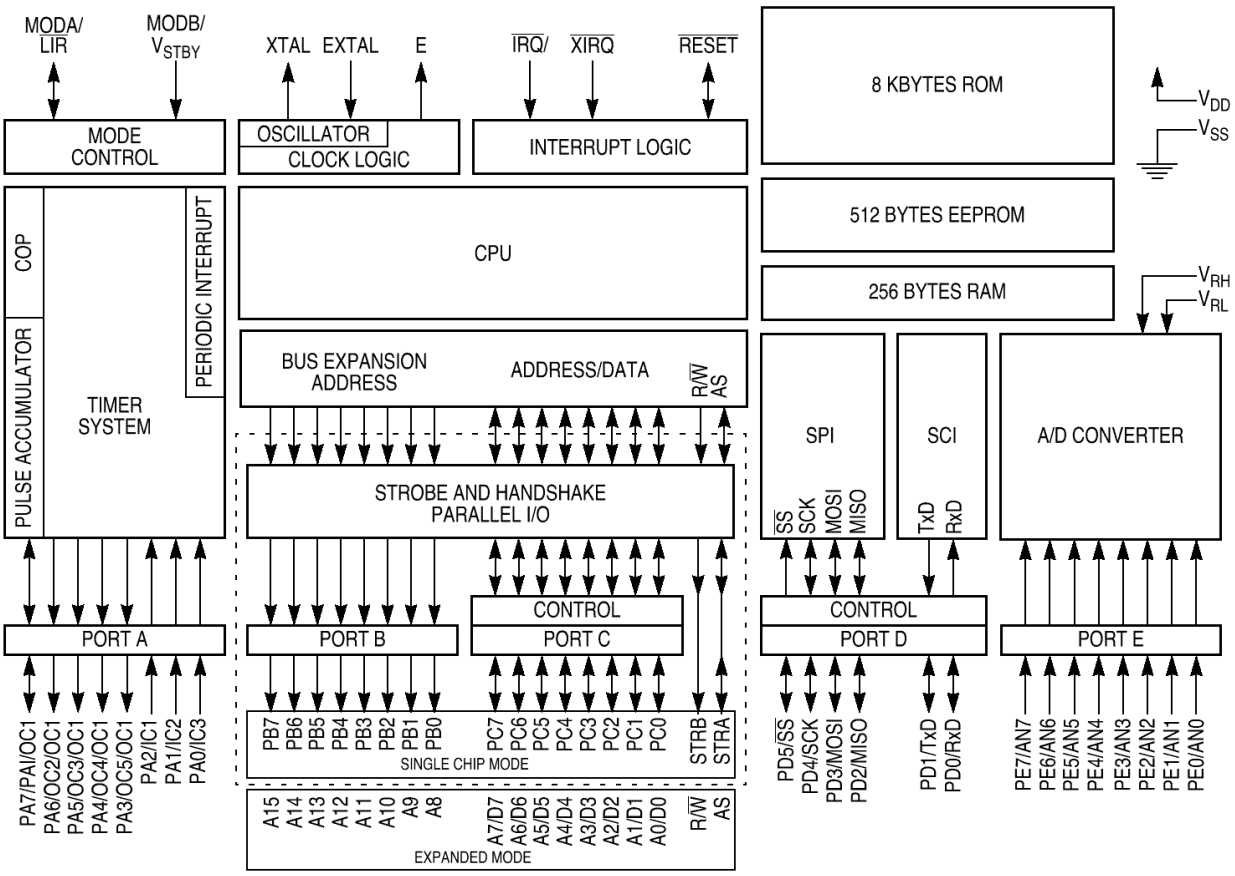


1) A partir du "Block diagram" détaillé du 8051, identifiez les fonctions intégrées dans le microcontrôleur (Horloge, Unité de control et de décodage des instructions CPU avec la gestion du compteur de programme PC, ALU, RAM, ROM/EPROM, Ports entrée sortie, Timers, et les registres à fonctions spéciales SFR (interface de communication série...)).

2) Localisez les bus de données et d'adresses et spécifiez leurs tailles (nombre de bits).

3) En déduire le type de structure (Von Neumann ou Harvard) du 8051 de chez INTEL.

<u>Exemple N°2</u>: Microcontrôleur MC68HC11 de MOTOROLA (doc MOTOROLA 11rm.pdf).

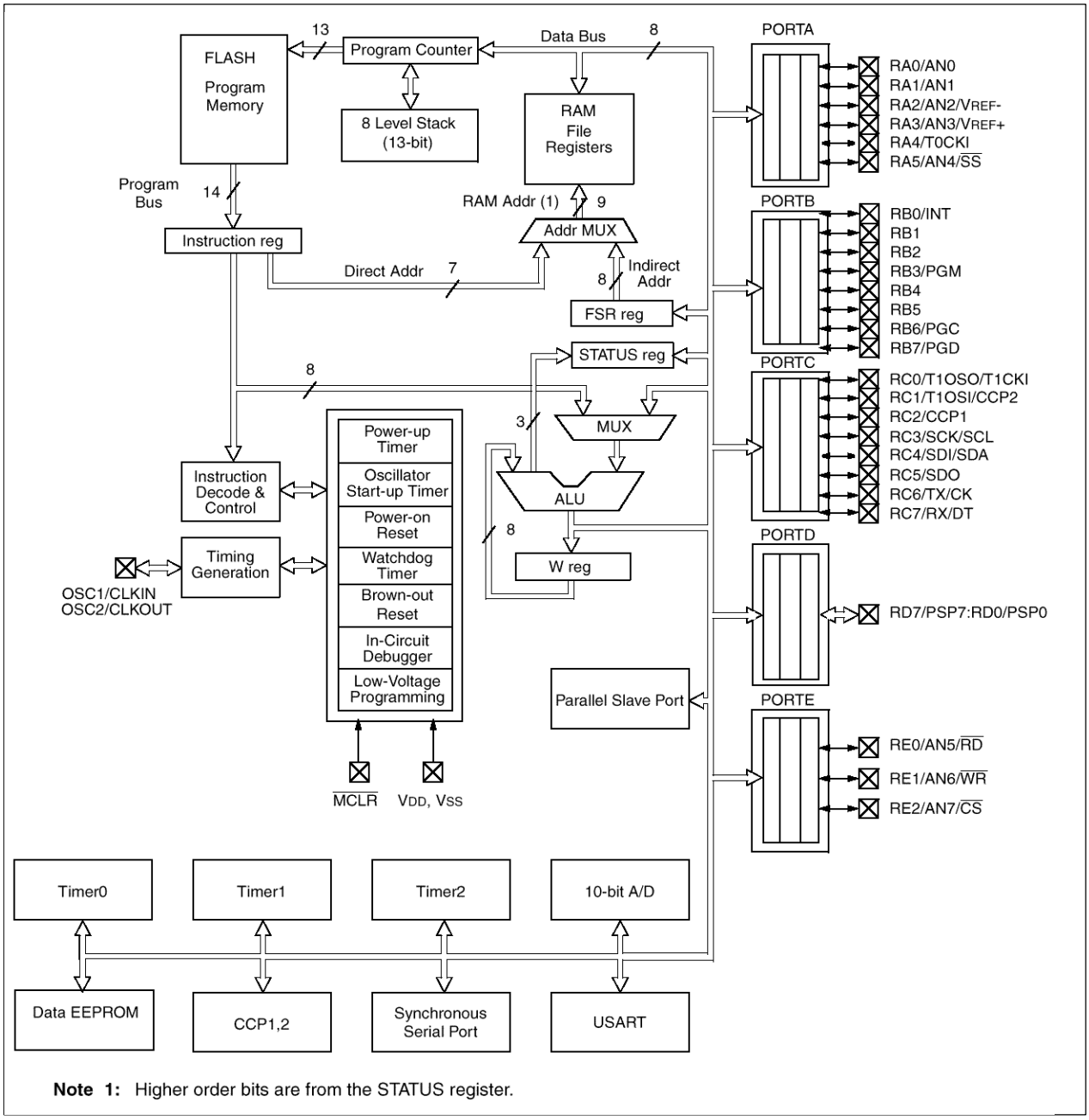


1) De même à partir du "Block diagram" détaillé du MC68HC11 de MOTOROLA, identifiez les fonctions intégrées dans le microcontrôleur (Horloge, Unité de control et de décodage des instructions CPU, RAM, ROM, EEPROM, Ports entrée sortie, Timers, Convertisseurs analogiques numériques, interface de communication série synchrone SPI et asynchrone SCI).

2) Indiquez pour chaque port (A à E) la direction possible des fils de port (entrée uniquement, sortie uniquement, ou bidirectionnel).

3) Comparez les caractéristiques fonctionnelles avec le 5051 de chez INTEL.

Exemple N°3: Microcontrôleur PIC16F877 de Microchip (doc Microchip 30292b.pdf).



1) A partir du "Block diagram" détaillé PIC16F877, identifiez les fonctions intégrées dans le microcontrôleur (Horloge, Unité de control et de décodage des instructions CPU, gestion du compteur de programme PC, ALU, RAM, Mémoire programme, EEPROM, Ports entrée sortie, Timers, interface de communication série synchrone et asynchrone USART, Capture/Compare/PWM (CCP) Module...).

2) Localisez les bus de données et d'adresses de la RAM et de la mémoire programme et spécifiez leurs tailles (nombre de bits).

3) En déduire le type de structure (Von Neumann ou Harvard) du PIC16F877 de chez Microchip.